An Analysis of Requirements Evolution in Open Source Projects: Recommendations for Issue Trackers

Petra Heck
Fontys University of Applied Sciences
Eindhoven, The Netherlands
p.heck@fontys.nl

Andy Zaidman
Delft University of Technology
Delft, The Netherlands
a.e.zaidman@tudelft.nl

ABSTRACT

While requirements for open source projects originate from a variety of sources like e.g. mailing lists or blogs, typically, they eventually end up as feature requests in an issue tracking system. When analyzing how these issue trackers are used for requirements evolution, we witnessed a high percentage of duplicates in a number of high-profile projects. Further investigation of six open source projects and their users led us to a number of important observations and a categorization of the root causes of these duplicates. Based on this, we propose a set of improvements for future issue tracking systems.

Categories and Subject Descriptors

D.2.1 [**Software Engineering**]: Requirements/Specifications; D.2.7 [**Software Engineering**]: Distribution, Maintenance, and Enhancement

General Terms

Management

Keywords

Open source, feature request, issue tracker

1. INTRODUCTION

Software evolution is an inevitable activity, as useful and successful software stimulates users to request new and improved features [29]. This process of continuous change of requirements is termed requirements evolution [16]. Requirements evolution management has become an important topic in both requirements engineering [16] and software evolution research [9].

Both industrial experience reports [24] and academic research have identified a significant set of software projects for which traditional notions of requirements engineering (RE) are neither appropriate nor useful [8]. In these settings, requirements still exist, but in forms different to what requirements textbooks typically characterize as best practice. These requirement approaches are characterized by the use of lightweight representations such as user stories, and a focus on evolutionary refinement. This is known as just-in-time RE [8].

This just-in-time RE is also found in open source projects [8, 18, 24]. Requirements in open source projects are managed through a variety of Web-based descriptions, that can be treated collectively as 'software informalisms' [24] . Traditional requirements engineering activities do not have first-class status as an assigned or recognized task within open software development communities. Despite the very substantial weakening of traditional ways of coordinating work, the results from open source software (OSS) development are often claimed to be equivalent, or even superior to software developed more traditionally [18].

Open source development has proven to be very successful in many instances and this has instigated us to explore how requirements are managed in open source projects. We expect to find a number of useful concepts that can be directly translated to more traditional software engineering trajectories as well, as these are slowly moving from the more traditional up-front requirements engineering to more agile RE [3].

In successful open source projects, many users get involved and start to request new features. The developers of the system receive all those feature requests and need to evaluate, analyze and reject or implement them. To minimize their workload it is important to make sure only valid feature requests are being entered. But the developers in open source projects have no control over what the remote users enter, so we need to analyze what happens at the side of the user that is entering the feature requests:

1. Is it easy for those users to see if the feature is already present in the system?
2. Is it easy to see if the same feature has already been requested by some other user?

Our idea is that by aiding the users in entering only new and unique requests, we can minimize the workload for developers that are maintaining the open source system. Our main research question is *how can we assist the users as the main actors in the requirements evolution process, with the purpose of simplifying the maintenance of the system.*

This leads us to our three subsidiary research questions:

RQ1 In what form do feature requests evolve in the open software community Web sites?

RQ2 Which difficulties can we observe for a user that wants

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
IWPSE'13, August 19-20, 2013, Saint Petersburg, Russia
Copyright 2013 ACM 978-1-4503-2311-6/13/08 ...$15.00.

to request some new functionality and needs to analyze if that functionality already exists or has been requested by somebody else before? Can we explain those difficulties?

RQ3 Do we see improvements to overcome those difficulties?

This paper describes an exploratory investigation into how requirements are managed in open source software projects. We witnessed difficulties that users have with entering only valid feature requests and we present recommendations for tool-support to overcome these difficulties. Our idea is that these recommendations can also be applied successfully in more traditional projects.

The structure of our paper is as follows: Section 2 describes the common structure of open source requirements. In Section 3 we introduce and analyze the problem of duplicate feature requests. In Section 4 we provide recommendations for avoiding duplicate requests. In Section 5 we present related work. In Section 6 we discuss our results before concluding in Section 7.

2. OPEN SOURCE REQUIREMENTS

Our first step is to investigate the different layouts of the open software *community web sites* in more detail. These websites are termed 'informalisms' by Scacchi [24] as they are socially lightweight mechanisms for managing, communicating, and coordinating globally dispersed knowledge about who did what, why, and how. We have analyzed several open source community web sites by browsing through their online repositories. We have zoomed in on those websites and analyzed the current state of requirements evolution. The open software community web sites that we have analyzed are listed in Table 1.

From that analysis we found a common structure in the requirements part. See Figure 1 for a generic overview of requirements related elements in open source project web sites. For each of those elements in Figure 1 we have analyzed what the relationship is with traditional requirements engineering, see our technical report [12].

In the web sites we have seen there is always some sort of 'ticket management system' for users to request new features for the system. Github has a very simple system (comments

Table 1: Projects analyzed

Apache Subversion	`subversion.apache.org`
Apache HTTPD	`httpd.apache.org`
Mozilla Firefox	`mozilla.org/firefox`
Mozilla Bugzilla	`bugzilla.org`
Android	`source.android.com`
Drupal	`drupal.org`
Tigris ArgoUML	`argouml.tigris.org`
Tigris TortoiseSVN	`tortoisesvn.tigris.org`
Netbeans	`netbeans.org`
Eclipse BIRT	`projects.eclipse.org/projects/birt`
Eclipse JDT	`.../projects/eclipse.jdt.core`
Eclipse MyLyn Tasks	`.../projects/mylyn.tasks`
Eclipse GMF	`.../projects/modeling.gmp.gmf-tooling`
KDE	`kde.org`
Gnome	`gnome.org`
Mono	`mono-project.com`
SourceForge	`sourceforge.net`
Google Code	`code.google.com`
GitHub	`github.com`
CodePlex	`codeplex.com`

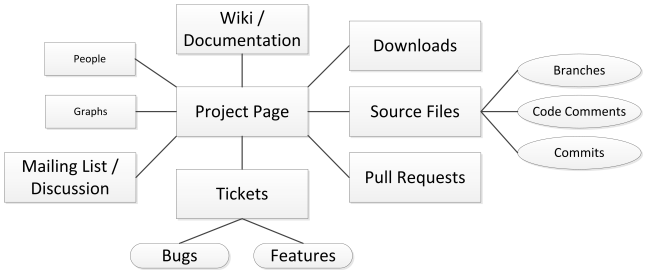

Figure 1: Open Source Requirements Items

that can be tagged) while Google Code's system is a bit more elaborate with type and status. Both sites include a voting system where users can vote for feature requests to emphasize their priority. The other sites use stand-alone issue tracker systems where the description in text is just one of the fields that need to be filled when requesting a new feature. Out of the sites we investigated, most of them use Bugzilla (a Mozilla open source project) as a ticket management system, see Figure 2. Note that Bugzilla is designed for managing defects ('bugs') so the way to indicate that this is a feature request is by setting the *Importance* to 'enhancement', although some projects (e.g. Apache Subversion) have a field called *Issue type* where 'enhancement' and/or 'feature' are values.

Well-organized open source projects require new requirements to be entered as a specific feature request, see for example `http://argouml.tigris.org/project_bugs.html` . The issue tracker is used to discuss the feature request, to assign the work to a developer and to keep track of the status of the request. Only in some smaller projects new features are implemented directly in (a branch of) the source code. Github uses *pull requests* to let users offer newly implemented features to the project (see `https://help.github.com/articles/using-pull-requests`) and automatically generates an issue in the tracker for each pull request to keep track of them.

To summarize we found that in most projects the requirements evolve in an issue tracker system.

3. DUPLICATE FEATURE REQUESTS

While analyzing the Eclipse JDT Core project we noticed the huge amount of duplicate feature requests (see Table 2). Our first step was to see if the same is true for the other 13 projects that use Bugzilla as an issue tracker [1]. We could easily track the duplicate requests by filtering on 'severity = enhancement' and 'resolution = DUPLICATE'. It turned out that many projects, including very mature and well-organized ones, have a high percentage of duplicate feature requests, see Table 2. The ones where the number of duplicates is lower, either have a strict policy (Apache HTTPD and Subversion warn the user explicitly to search/discuss before entering new issues), are smaller (Eclipse MyLyn and GMF) or have a company behind the project (Mono and Android). One can easily argue that with the large number of issues in the database not even all duplicates have been marked.

So apparently users cannot or do not find out if the feature they request is really new or has already been implemented

[1]Mono has a split Bugzilla repository since the project went from Novell to Xamarin

Table 2: Duplicate Feature Requests in Open Source Projects Jan 2012

Project	# Duplicate	# Request	%
Apache HTTPD	28	809	3
Apache Subversion	66	881	7
Mozilla Firefox	2920	8023	36
Mozilla Bugzilla	1166	5664	21
Android	283	5963	5
Drupal	1351	7855	17
Tigris ArgoUML	133	562	24
Netbeans	2896	10711	27
Eclipse MyLyn Tasks	16	403	4
Eclipse GMF	17	370	5
Eclipse JDT	1551	8314	19
GNOME Evolution	1843	6895	27
Mono Xamarin	11	477	2
Mono Novell	81	5277	2

or requested before. Our next question was: what is the reason for those duplicates?

3.1 Research Strategy

The strategy we chose is to look at one of the projects with a strict policy (Apache HTTPD) and to see why still those 28 duplicates were reported. Out of the 28 duplicates one turned out to be a defect, not an enhancement. For each of the 27 remaining duplicates we analyzed the history of comments and tried to identify what the root cause for this duplicate was. We did this by answering the following questions:

- Is it a real duplicate? In other words: is the request of the author indeed the same as the original request?
- Could the author of the request have found the duplicate by searching in the issue tracker? With what search terms?
- Did the author submit some source code directly with the request?
- Was the duplicate submitted long after the original request?
- Who marked the duplicate? Was there an overlap in people involved in original request and duplicate?
- Do we see any way in which the author could have avoided the duplicate entry?

Figure 2: Feature Request in Bugzilla

The analysis was done manually by the first author by reading the title and comments for each duplicate and analyzing what caused the reporter of the feature request to send in the duplicate; could the reporter have avoided that? After concluding this analysis for the Apache HTTPD project we had an initial categorization of duplicates by root cause.

Subsequently, we repeated the analysis for 5 other projects to validate the findings. These projects are Subversion, Firefox, Netbeans, Eclipse JDT and Novell Mono. Again this manual analysis was done by the first author. While we analyzed all duplicates for Apache HTTPD, the other projects had many more duplicates (see Table 2), so we had to select samples. For each of the projects we selected the 30 most recently reported duplicates between 01 Jan 2011 and 31 Dec 2012. For Mono Novell we just selected the 30 most recently reported duplicates without a time window (because of the smaller number of total duplicates).

This second round of manual analysis led us to do a slight adjustment to the initial categorization. The author category is now not only used for authors that enter the same request twice within a few minutes (we only saw this in the HTTPD project) but also for other situations where the author seems to be aware of the duplicate he/she created. We initially had a specific category for feature requests being duplicates of defects, but in other projects we saw situations where the *product* attribute did not match. We decided to group those two situations into 1 category for 'mismatch of attributes'.

Next to the detection of distinct categories of duplicates we did a number of other interesting observations which we discuss below.

3.2 Categorization

Grouping the duplicates with similar root causes resulted in the following categories:

Duplicate Solution [DS] This is not a real duplicate request. The request of the author is new but happens to have the same solution as a request that was posted before.

Partial Match [PM] Only part of the request has been posted before; the main questions for original and duplicate are completely different.

Patch [PA] The author has submitted a patch directly with the request. Our assumption is that the fact that the author is proud of his own code makes him/her lazy in checking if the same has already been done before.

Author [AU] The same author enters his/her request twice or indicates in title or comments that he/she is aware of the duplicate entry.

Mismatch Attributes [MA] The original request has different values for important attributes (in our observations: *defect type/importance* or *product*) so the author

Table 3: Analysis of Duplicate Feature Requests

	HTTPD	Subversion	Firefox	NetBeans	Eclipse JDT	Mono Novell	#	%
Explicit warning	Y	Y	N	N	N	N		
Solution [DS]	3	5	3	2	0	4	**17**	**10**
Partial [PM]	1	2	2	0	2	2	**9**	**5**
Patch [PA]	10	1	0	0	0	0	**11**	**6**
Author [AU]	4	5	0	2	2	7	**20**	**11**
Mismatch [MA]	1	0	20	3	7	4	**35**	**20**
Wording [WO]	7	6	2	5	5	3	**28**	**16**
No Check [NC]	1	11	3	16	14	10	**55**	**31**
(No duplicate)	0	0	0	2	0	0	**2**	**1**
#	**27**	**30**	**30**	**30**	**30**	**30**	**175**	**100**

might not have searched with the correct search values. An author that is entering an 'enhancement' might not think to include 'defects' in the search for existing duplicates. An author that is entering a feature request for the *product* 'Firefox' might not have included the 'Core' *product* in the search for existing duplicates.

Wording [WO] Different wording for the same problem is used in both original request and duplicate. Possibly the author did not detect the duplicate because he/she was not searching with the right terms.

No Check Done [NC] It is not clear why the author did not detect the duplicate while the duplication is easy to spot with a simple search.

For a complete analysis of the Apache HTTPD project see our technical report [12]. Table 3 indicates for each of the analyzed projects the number of duplicates in each category.

Note that for the NetBeans projects two duplicates out of 30 ([216335] and [217150]) turned out to be no real duplicates and thus do not fall into any of the categories. In [216335] the 'DUPLICATE' link is misused to indicate some other type of relation (see **O8** below). In [217150] one person marks it as a duplicate (*"looks like another manifestation of a problem introduced in JavaFX SDK 2.2 by, as described in issue #216452"*) but two other persons later agree that this is not true.

As can be seen in Table 3 each category was present in at least two projects. Not each project shows the same division over the categories. A project like HTTPD has many duplicates because of patches added to the feature request, where as in other projects we did not find any patches. The FireFox project shows many problems with mismatching attributes because Mozilla uses one big Bugzilla database for all its Mozilla projects. FireFox is just one of the products in the database and thus it is easy to select the wrong product when entering a new request or to search for the wrong product when checking for duplicates. A bigger project with a wide user base like NetBeans shows more duplicates in the category NC. We can assume that reporters of new feature requests get easily discouraged searching for duplicates by the huge volume of existing requests. Furthermore NetBeans does not explicitly warn users to check first before entering new requests, as opposed to HTTPD.

The two Apache projects include an explicit warning at the beginning of the process, see Figure 3. The projects that are marked in Table 3 as 'N' under 'Explicit warning' do not have such an explicit warning at the beginning of the process. All of them however include a small notice or an optional step to search for previously entered issues or to ask on the discussion list first, but this is less compelling for the user than an explicit warning.

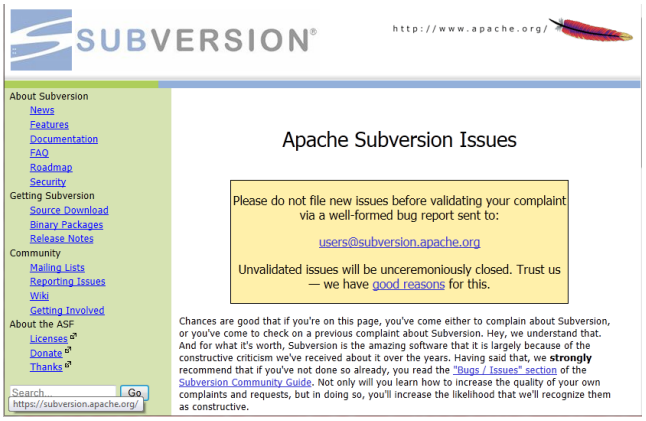

Figure 3: Warning on the bug report page of Subversion

To summarize we can state that users do have difficulties in submitting unique feature requests, for different reasons. For each of these root causes we would like to offer support to the user to avoid duplicate requests.

3.3 Further Observations

While analyzing the six projects for duplicate feature request we did not only categorize the duplicates but also made some interesting observations:

[O1] Many of the 14 projects in Table 2 have a high percentage of duplicate feature requests. It seems to be the case that a project such as Apache HTTPD that explicitly warns the user to search and discuss before entering new requests can greatly reduce the number of duplicates. The Subversion project goes even further and explicitly asks users to file a bug report at a separate email address to get it validated before posting in Bugzilla, see Figure 3.

[O2] Given the high number of feature requests a manual search for certain terms can easily yield a high number of results. In our experiments it sometimes took a sophisticated search query with enough discriminating attributes set to obtain a set of search results that is small enough to manually traverse them looking for a duplicate. Even for us (who knew the duplicate we were looking for beforehand) it often took more than one step to come to the correct search query. This involved stemming of the search terms (e.g. "brows" instead of "browsing") and searching in comments of the issue instead of the summary. From our own experience we can say that the simple search screen (shown by default by Bugzilla) is not enough; for most issues we needed to open the advanced search screen to find the duplicates.

[O3] Some feature requests were marked as a duplicate of a defect. Often there was discussion in the comments of an issue about whether the issues should be classified as a defect or an enhancement. Apparently this difference is not always easy to make for a user entering a new request [13]. Projects like Subversion make it even more complex by adding 'enhancement' and 'fea-

ture' both as separate types. Is it a completely new feature or an enhancement to an existing one? With a broad existing product like Subversion this question is extremely difficult to answer. One could argue that everything is an extension to the version control system and thus an enhancement. The risk is that users entering new requests will not search for duplicates of the type 'defect' and thus not find potential duplicates.

[O4] Marking of the duplicates within one project is done by a wide variety of project members: e.g. in Apache HTTPD the 27 duplicates were marked by 18 different user names. The users that marked the duplicates in this case were also not all of them part of the HTTPD core team. When we check the activity of those users in Bugzilla, we see that they are involved in 5 to 1338 issues (as Commenter or Reporter), with about half of the 'markers' involved in more than 400 issues and the other half involved in less than 75. This observation tells us that we can not assume that duplicates are easily filtered out by a select group of developers that scans all new requests.

[O5] The time gap between the duplicate and the original request is arbitrary. In the Apache HTTPD project this ranged from 0 to 88 months. We expected to see short time gaps because we expected the user needs to be influenced by time-bounded external factors (e.g. the emergence of a new standard that needs to be applied), but this turned out to not be the case.
Note that the time gap between creation of the request and marking it as a duplicate is also arbitrary. In the Apache HTTPD project about half of the requests were marked within 2 months but the longest time gap for marking the duplicate reached up to 56 months. This indicates that some duplicates stay undetected for a long time.

[O6] During the manual analysis of the duplicates we were often hindered by the fact that many issues include a lot of off-topic comments: comments that do not pertain to the issue itself but, e.g. to project management topics or for social interaction. The same problem was detected by Bettenburg et al. [1] for bug reports in general. Examples for feature requests in Apache Subversion:

> [Issue 3415] *This would be a very useful addition to the product IMO.*
>
> [Issue 3030] *I see. Good to know that the issue has will be resolved in the next release. I understand you suggestion about the mailing list - however joining a mailing list for one issue in 3 years seem an overkill. (Be proud: I use Subversion ever day and had only one problem which nagged me enough to raise a report) Personally I think Google-groups got it right here: they have a read/write web interface for casual users and mail distribution for heavy users.*
>
> [Issue 2718] *Except that that bug report has undergone the usual open source 'me too'/'let's add some irrelevant technical detail' treatment, which made me create a clear and concise bug report that someone could actually fix in the next 12 months.*

Having such useless comments in the issues makes it more difficult for a user or developer to quickly grasp the content of the issue and thus more difficult for the user to see if the issue is a duplicate of the one he/she is about to enter.

[O7] We saw cases of larger structures of master-duplicate relationships such as the example from Subversion in Figure 4. In this example many issues around 'improved error messages' are linked together. Currently the user exploring such structures only has the possibility to click on each of the links one by one and build such a graph in his/her head. The risk is that the user will get lost while doing this for larger structures.

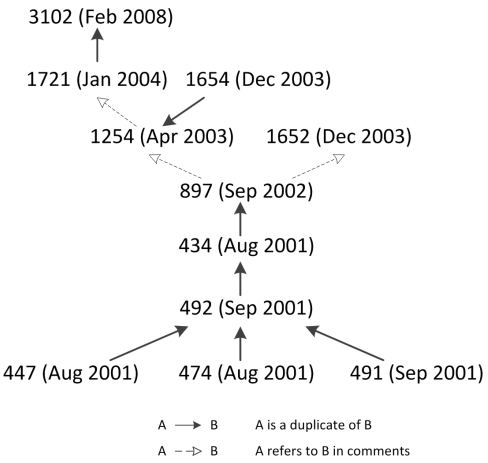

Figure 4: Network of 'duplicate' relations in Subversion

Note also that in this entire list, only issue 434 is typed as an 'enhancement', the others are typed as 'defect', 'task' or 'patch'. This is related to observation **O3**.

[O8] For the moment Bugzilla has only two explicit linking mechanisms: 'request A blocks request B' and 'request A is a duplicate of request B'. Only the first type can be visualized in a dependency tree. Another option is to implicitly link requests by entering comments with the ID of the other request included [23]. This limitation of Bugzilla leads to misuse of the 'DUPLICATE' link as we saw already for the categories DS and PM. This misuse makes it difficult for users and developers to see the real link between two requests, which hinders comprehension of the work done or to be done.
An example of this misuse is in NetBeans issue [216335]:

> *Although it is not exactly the same idea I am marking this as duplicate of issue #208358 as it addresses the same problem. Having the two enhancements logged in one place makes sense due to their close relation.*

[O9] Most users that request new features are developers themselves, as Noll states that the majority of requirements are asserted by developers [19, 20]. Developers implement what they need themselves: this will lead to on average the mostly needed features. In practice this means that part of the 'feature requests' in the issue tracker are already accompanied by a piece of source code in which the user that requests the fea-

ture implements the feature (see also **PA** before). The feature requests with patches included are often not well-described (the documentation is the source code). This makes it difficult to analyze the feature request when looking for duplicates.

All of these observations lead us to believe that things could be improved for the projects we have investigated and that those improvements could also benefit other projects. That is the topic of the next section.

4. ASSISTING USERS TO AVOID DUPLICATE REQUESTS

Issue trackers like Bugzilla are designed as bug trackers and thus not a perfect tool for managing requirements. Some fields (e.g. rationale, source) are missing [14, 21] leading to information loss already at the entry of a new request. To see how many users need a new feature a voting mechanism is one of the few online instruments [7], but this is not always present. The main problem however is that it is difficult to indicate hierarchy or relations between requirements and to get an overview of relations that *have* been defined [23]. This leads again to information loss (links that were known at request creation time are lost) and makes it difficult to get an overview of existing feature requests later on. Another problem is that the commenting mechanism is difficult for maintaining requirements while a comment is a free-text field. The user can enter anything he/she wants and Bugzilla enters some auto-comments for certain actions (e.g. marking as a duplicate). For the reader it will be difficult to get a quick overview of feature requests with many comments (What is the type of each of the comments? What is the final decision on the request?). Despite these disadvantages the usage of such issue trackers remains high because of the advantage of managing all development tasks (fixing bugs, writing documentation, implementing new features, ...) in one single tool.

With those drawbacks in mind we investigate the observations done before and come up with some recommendations to improve the system for the user. We start with recommendations for manual search and issue creation and end with implications for extended tool support. For each of the items we refer to the category of duplicates (DS, PM, ... , NC, see Section 3.2) or the observation (O1 till O8, see Section 3.3) that has lead us to the recommendation.

4.1 Recommendations for Manual Search and Creation

[**R1**] Users that search for duplicates should include both defects and enhancement types in their queries (**O3**, **MA**).

[**R2**] Users that search for duplicates can not exclude issues before a certain date (**O5**). They could only do that if they know their request (e.g. implement a new standard) has an explicit date limit (nobody could have asked for the standard 1 year ago because it only exists for half a year).

[**R3**] Projects should include warnings to search first and to ask on mailing or discussion lists before entering a new request (**O1**, **NC**). Research has shown that most duplicates are reported by infrequent users [5] so giving them a reminder of the procedure or explicit instructions could help filter out some of the duplicates. Furthermore, when the user submits a request that includes a patch an explicit warning should be repeated to remind the user that he/she should search for similar solutions first (**PA**).

[**R4**] Projects should include a link to clear guidelines on how to enter issues (e.g. when is it a defect or an enhancement) to ensure that all fields are filled correctly and to avoid users entering new requests for new versions of the software (**AU**, **MA**)

[**R5**] Users entering new feature requests should only include issue-related comments; the same holds for the users commenting on existing feature requests (**O6**). For other types of comments the mailing/discussion list should be used (from where the user can easily hyper-link to the request). Projects could even go as far as removing unrelated comments from the request, to keep a 'clean' database.

[**R6**] Users should be told to split feature requests into atomic parts: one request per issue ID (**PM**). This makes it easier later on to follow up on the request and link other requests to it. When developers looking at the issue see that it is not atomic, they should close it and open two or more new ones that *are* atomic. The partial match example of Apache HTTPD shows that certain wordings can hint at non-atomic requests:

> *[29260 - Author] The base functionality of mod_vhost_alias for dynamic mass virtual hosting seems to work great, but there are* **a couple things** *that need to be changed to work with it.*
> ...
> *[29260 - Marker] Most of this is WONTFIX, but the stripping www thing is reported elsewhere.*
> **** This bug has been marked as a duplicate of 40441 ****

[**R7**] Projects should not accept patches or other source code as feature requests (**PA**). A patch is not a feature request, it is a request from the author that asks the project to look at something already implemented by the author. A mechanism like the pull request that GitHub (`https://github.com/`) uses is much more appropriate for submitting patches. In that way the patch is immediately available to all users (it sits waiting as a branch of the trunk repository until the owner of the trunk accepts and merges it) and this avoids users to enter the same patch twice. At the least a project could create a separate issue type for patches, making it easier to search for similar patches submitted earlier.

[**R8**] Projects should make clear what the hierarchy of products or components is within their issue database (**MA**). A user searching for duplicates should also include parent, children or siblings of the product he/she intended to search for, because a duplicate feature request might have been added for one of those. Especially for novice users the structure of the (group of) products might not be clear. This means they will not find some duplicate requests and unnecessarily enter a new request.

4.2 Implications for Tool Support

The fact that in the investigated Apache HTTPD project

many different developers are involved in marking the duplicates (**O4**) and many different users are involved in entering requests indicates that we need some kind of tool support. Within such open source projects we can simply not rely on one small group of users or developers to keep an overview of the project and filter out inconsistencies.

We would like to prevent problems as early as possible: at the moment the user is entering new feature requests. Based on our observations we see different opportunities for extended tool support:

Synonyms

The search tool could automatically suggest alternative search terms based on a list of synonyms or closely related words for the project, e.g. '.jpg' and 'image' (**WO**). This list could be compiled by language processing the existing feature requests and clustering them or could be manually composed. Each time a new duplicate request is found the developer marking the duplicate could verify if it was caused by a wording problem. If so, the different wordings could be added to the synonym list.

Duplicate Submission

The fact that a single author submits his/her request twice within a few minutes (we saw examples of this in the HTTPD project) could easily be filtered out automatically (**AU**). After submission, it should first be checked if the same author has already submitted a request with the same summary line, before the request is saved in the database.

My Requests

In the Subversion project we saw an example of an author 'forgetting' an earlier submitted request:

> *Wow. I *completely* forgot that I'd implemented this already (per issue #3748). Suddenly, I feel old.*

But there are more cases of the author being aware of some form of duplication (**AU**). It would be good that when a user logs in to submit a new request, he/she is first presented with a list of his/her previously submitted requests. This can serve as a reminder and as a status check. Instead of entering a new request the user can more easily go to one of the existing requests and extend it with additional information.

Comments

The issue reporting tool should offer some advanced support for discerning the different types of comments in a feature request (**O6**). In the examples we looked at, we found comments on test cases, procedures, questions, answers to questions in other comments, automatic actions, special tags to search for a set of related issues, complaints, discussions, etc. It is not clear what type of comment it is and thus what the relevance is without reading the whole comment. And sometimes it is difficult to see which comment is answering which other comment. If there would be separate fields for some of the types (e.g. auto-generated comments could be displayed separately in some sort of history section) or tags/colors related to the type of comment this would greatly simplify the ability of a user to get a quick overview of a feature request. This overview is needed to judge if the request is a duplicate or not of the one the user is about to enter. Of course this would also demand from

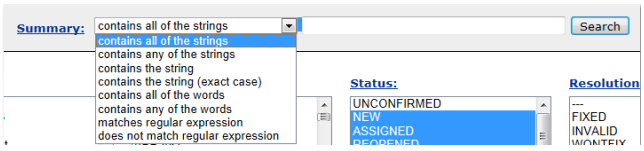

Figure 5: Search options in Apache HTTPD Bugzilla

the users entering the comments that they would use the correct tag/color for their comment, but this seems like a small effort which yields a high benefit.

Linking

We need some more sophisticated linking mechanism between feature requests than the current ones in Bugzilla (**O8**, **DS**, **R6**). We could imagine newly added fields in a feature request where the user or developer can indicate links with a type and link comment, e.g. a "solution" link with comment "both can use the same library with mathematical functions mathlib". This would avoid misuse of the 'DUPLICATE' link and would keep users from entering comments that merely serve to indicate an informal link between requests. The extended linkage information can be useful for newcomers to get a grasp of the project.

Visualization

Currently Bugzilla only offers to visualize the so-called 'dependency tree'. This is a graph showing which issues 'block' each other. This 'blocking' must have been indicated before by the user. For feature requests it would be more useful to be able to visualize the duplicate relations, as we did in Figure 4 (**O7**). And also other links if implemented (**Linking**) are a good candidate to visualize in such graph structures. We can even think of implementing tools that automatically process the feature requests to infer links between them from their texts and then visualize them.

Advanced Search

Last but not least we can think of more intelligent search engines for feature requests (**O2**, **R1**). The Bugzilla implementation of the HTTPD project searches on strings and regular expressions, see Figure 5. This is a good start but we suspect that natural language based search can greatly improve the search results. The search tool could be extended to automatically include the related products or components in the search (**R8**). Also the presentation of the search results to the user could be improved, e.g. with clustering or visualization techniques. A first attempt for better presentation of search results was made by Cavalcanti et al. [4] by highlighting search terms in the result set.

To summarize we see many opportunities for advanced tool support. All of these functionalities will help in the process of understanding the current set of feature requests in a project. This serves to avoid duplicates but also in other situations where the user needs to get an overview of the existing feature set of a system, e.g. when documentation of the system's functionality is missing. What remains is to actually build those tools and validate them in open source and company-based projects.

5. RELATED WORK

5.1 Requirements Evolution

There are many papers on the analysis of open source projects, but not so many cover requirements engineering topics. One of the first overview papers has been written by Scacchi [24]. The paper describes the open source requirements engineering process and the form of requirements in open source projects. Our analysis is based on the one of Scacchi, but proceeds to the next level of detail. For Scacchi 'open software web sites' is just one of the forms of requirements. In our analysis we dive into this by subdividing it into many different parts of requirements, see Figure 1.

5.2 Duplicate Detection

The fact that much duplication exists in the requirements of open source projects has also been detected by Clelland-Huang et al. [6]. In their research they focus on open forums, not on issue trackers. For the requirements-related messages on these open forums they propose an automatic clustering technique, which we could also apply in future work on feature requests in the issue tracker.

Gu et al. [10] use a similar clustering technique to automatically suggest duplicate issue reports to the author of a new issue report. Their recall rate is between 66 and 100%. Runeson et al. [22] achieve a recall rate around 40% when analyzing defect reports using Natural Language Processing (NLP) with a vector-space-model and the cosine measure. Sun et al. [25] claim they obtain significantly better results by using a discriminative model. Wang et al. [28] do not only use natural language processing but also use execution information to detect duplicates.

In our analysis we found an explanation why Gu and Runeson do not detect all duplicates: not all issues marked as 'Duplicate' are real duplicates in the sense that they could have been detected with natural language processing. This leads us to believe that next to experimenting with clustering as in [6] and [10], we need some more sophisticated techniques like e.g. visualization, to support the author in getting an overview of the feature requests posted before.

Tian et al. [26] extend the pioneer work of Jalbert and Weimer [15] to improve the method of bug classification: is the bug a unique one or a duplicate? This classification could of course help in warning users that they might be submitting a duplicate, but considers only one aspect of the problem.

A different approach is used by Vlas and Robinson [27] who have developed an automated Natural Language requirements classifier for open source projects. The tool classifies the individual sentences of a feature request into types of requirements according to an extended McCall model [17], e.g. 'operability' or 'modularity', with a pattern-based approach. A similar classification could also help in clustering complete feature requests, as we are looking for.

Cavalcanti et al. [4] present a tool called BAST (Bug-reports Analysis and Search Tool) that provides an extended keyword-based search screen for issues. It shows issues with the same keywords, but additionally in the lower part of the screen a quick overview of important issue attributes. Their study shows that it worked better than the normal search tool for one company. The main drawback of their tool is that it is still based on keyword search and thus depends on the user entering the correct search terms. This contrasts our idea that synonym-based search should also be implemented. In a further paper Cavalcanti et al. [5] explore the duplication of bug reports in several projects and come up with recommendations. Most of those recommendations also match ours as they are also valid for feature requests. However, our paper adds some feature request specific recommendations like the handling of patches, the improved linking mechanism and a better separation of comment types.

Where Bettenburg et al. [2] claim that duplicate bug reports are not always harmful, the same can be true for feature requests: two feature requests that are similar can help the developer in understanding the requirements. However this requires that the developer is aware of the duplicate before starting on the implementation. An unnoticed duplicate feature request can easily lead to two different developers implementing the same feature in different places in the system. This strengthens our claim that duplicate feature requests should be detected/prevented early on.

Our approach differs from the ones mentioned in this subsection because we focus on feature requests only. Feature requests, or requirements, are different from defects. They require different initial content (what the user needs and why vs. what is not working) and have different life-cycles. A defect stops living once resolved, but the description of a requirement is still valid documentation once implemented. We expect to extend or detail the approaches mentioned above with some requirements-specific analysis. With that we are not so much interested in the automatic detection of duplicates, but in supporting the user to get a better overview of the existing feature requests such that the user can more easily see which related (not necessarily duplicate) feature requests have already been submitted.

5.3 Visualization

Sandusky et al. [23] studied what they call 'Bug Report Networks (BRN)' in one open source project. This BRN is similar to what we have drawn in Figure 4. In the bug report repository they studied 65% of the bug reports sampled are part of a BRN. They conclude that BRNs are a common and powerful means for structuring information and activity that have not yet been the subject of concerted research by the software engineering community. The continuation of this stream of research will result in a more complete understanding of the contribution BRNs make to effective software problem management. We support this from our own findings and plan to investigate what would be the best way to visualize the BRN's for our specific application domain of feature requests.

6. DISCUSSION AND FUTURE WORK

This section discusses the results and describes opportunities for future work.

The Research Questions Revisited

RQ1 *In what form do feature requests evolve in the open software community Web sites?* We analyzed the community web sites of a number of open source software projects and we found that while requirements are sometimes discussed on discussion fora or mailing lists, they are typically channeled towards the issue tracker. In particular, we observed that many open source web sites use Bugzilla as a requirement management tool

to support requirements evolution.

RQ2 *Which difficulties can we observe for a user that wants to request some new functionality and needs to analyze if that functionality already exists or has been requested by somebody else before? Can we explain those difficulties?* We found that many duplicate feature requests exist within the projects. This indicates difficulties that user have to submit unique feature requests. We have categorized the duplicates by root cause; we created seven categories of duplicates which we have observed in six projects.

RQ3 *Do we see improvements to overcome those difficulties?* By analyzing the root causes of the duplicates we have suggested improvements and tool support to avoid duplicates, e.g. improved linking mechanisms, visualization of those links, clustering or advanced search. All of these functionalities will help in understanding the current set of feature requests in a project. This serves to avoid duplicates but also in other situations where the user needs to get an overview of the existing feature set of a system, e.g. when documentation of the system's functionality is missing.

In future work we plan to investigate the tool support implications to see if indeed we can improve the requirements evolution of projects by providing extended tools.

Validity

We did not verify our assumptions by contacting the original authors or interviewing developers in other open source projects to see if they recognize the problems with duplicate requests. Based on the anecdotal evidence that we gathered from analyzing the issue tracker, we believe that projects would benefit from extra tools to get an overview of all feature requests but this must be validated in our future work.

We are also aware of the fact that the issue tracker must stay a tool that users still want to use to report new feature requests. This means that we can not include to many obstacles for new users to enter requests. Practical experiments must be done to validate what is 'too many'.

Applicability

We have conducted our case study on an open source project. Also in companies there will be situations where end users have direct access to the issue tracker tool to enter new feature requests, so that the problem of duplicate entries is relevant. Furthermore our tool support will also help newcomer developers to get an overview of the project. In the above we have claimed that the results are also valid for company-based projects. In our future work we plan to validate this claim by applying the methods we develop also on company-based projects.

Issue Trackers

We have done a small comparison with two other widely-used issue trackers, namely *Jira* (`www.atlassian.com/JIRA`) and *Trac* (`trac.edgewall.org`), see Table 4. This shows us that the recommendations we have for Bugzilla are also valid for other issue trackers. The only striking difference is the fact that Jira offers a free linking mechanism (like we recommended in 'Linking' in Section 4.2). However a newer tool like Trac does not offer this, so the recommendation in general is still valid. And Jira also does not offer any visualization of those advanced links.

Table 4: Comparison with Other Issue Trackers

	Bugzilla	**Jira**	**Trac**
Launch	1998	2003	2006
Custom Fields	In UI	In UI	In DB and config file
Labeling	With keywords	With labels	With keywords
Link	'Blocks' and 'Duplicate'	'Blocks', 'Duplicate' and 'Relates to' with link comment	'Duplicate'
Voting	Yes	Yes	No
Search	Built-in engine searches for keywords, substrings; simple and advanced search UI	Lucene engine; whole words only but can be told to stem words or do 'fuzzy' search; simple and advanced search UI;	Built-in engine searches for keywords and substrings; advanced search done with queries
Extensibility	Plugins & Open Source	Plugins	Plugins & Open Source
Interfaces	XML-RPC, REST	REST, Java API	XML-RPC

Other Questions

High quality feature requests could simplify the evolution of the system. But how do we define the quality of those feature requests? For regular requirements there are many characteristics that qualify a good requirement (e.g. atomic, no ambiguity, prioritized) [11] but do they all hold for feature requests in an issue tracker such as Bugzilla? Can we observe interesting facts on the quality of feature requests? Do we see ways to improve the quality of feature requests? Bettenburg et al. [1] did similar work (including a tool) for bug reports in general, but not all their results are valid for feature requests.

7. CONCLUSION

In this paper we have investigated requirements evolution in open source project web sites and saw that in most projects an issue tracker is used to evolve requirements. Within those web sites that use Bugzilla as a requirements management tool we have observed a high number of duplicate feature requests. We have made a classification of the causes for these duplicate feature requests. Using this classification we have given recommendations and implications for tool support to avoid duplicate feature requests.

Our main goal for future work is to improve tool support for dealing with feature requests in issue trackers. An important step in this direction is to give the users of these issue trackers an overview of the project, including relationships between already existing feature requests. Better search facilities and a hierarchical exploration of requirements are subsequent steps towards mechanisms to prevent duplicate feature requests from being entered. Our proposed tools will also benefit company-based projects, since a lot of them use Bugzilla-like tools for managing requirements evolution.

8. ACKNOWLEDGMENTS

This work has been sponsored by the RAAK-PRO program under grants of the EQuA-project.

9. REFERENCES

[1] N. Bettenburg, S. Just, A. Schröter, C. Weiss, R. Premraj, and T. Zimmermann. What makes a good bug report? In *Proc. Int'l Symposium on Foundations of Software Engineering (FSE)*, pages 308–318. ACM, 2008.

[2] N. Bettenburg, R. Premraj, T. Zimmermann, and S. Kim. Duplicate bug reports considered harmful ... really? In *Proc. Int'l Conf. on Software Maintenance (ICSM)*, pages 337–345. IEEE, 2008.

[3] L. Cao and B. Ramesh. Agile requirements engineering practices: An empirical study. *IEEE Software*, 25(1):60–67, 2008.

[4] Y. C. Cavalcanti, C. E. A. da Cunha, E. S. de Almeida, and S. R. de Lemos Meira. Bast - a tool for bug report analysis and search. In *XXIII Simpósio Brasileiro de Engenharia de Software (SBES)*, Fortaleza, Brazil, 2009.

[5] Y. C. Cavalcanti, P. A. da Mota Silveira Neto, D. Lucrédio, T. Vale, E. S. de Almeida, and S. R. de Lemos Meira. The bug report duplication problem: an exploratory study. *Software Quality Journal*, 21(1):39–66, 2013.

[6] J. Cleland-Huang, H. Dumitru, C. Duan, and C. Castro-Herrera. Automated support for managing feature requests in open forums. *Commun. ACM*, 52(10):68–74, 2009.

[7] J.-M. Dalle and M. den Besten. Voting for bugs in firefox: A voice for mom and dad? In *OSS*, volume 319 of *IFIP Advances in Information and Communication Technology*, pages 73–84. Springer, 2010.

[8] N. Ernst and G. Murphy. Case studies in just-in-time requirements analysis. In *Int'l Workshop on Empirical Requirements Engineering (EmpiRE)*, pages 25–32. IEEE, 2012.

[9] N. A. Ernst, J. Mylopoulos, and Y. Wang. Requirements evolution and what (research) to do about it. In *Design Requirements Engineering: A Ten-Year Perspective*, volume 14 of *LNBIP*, pages 186–214. Springer, 2009.

[10] H. Gu, L. Zhao, and C. Shu. Analysis of duplicate issue reports for issue tracking system. In *Int'l Conf on Data Mining and Intelligent Information Technology Applications (ICMiA)*, pages 86–91, 2011.

[11] P. Heck and P. Parviainen. Experiences on analysis of requirements quality. In *Int'l Conf. on Software Engineering Advances (ICSEA)*, pages 367–372. IEEE, 2008.

[12] P. Heck and A. Zaidman. An analysis of requirements evolution in open source projects: Recommendations for issue trackers. Technical Report TUD-SERG-2013-007, Software Engineering Research Group, Delft University of Technology. `http://swerl.tudelft.nl/bin/view/Main/TechnicalReports`.

[13] K. Herzig, S. Just, and A. Zeller. It's not a bug, it's a feature: How misclassification impacts bug prediction. Technical report, Universitaet des Saarlandes, Saarbruecken, Germany, August 2012.

[14] IIBA. A guide to the business analysis body of knowledge (babok guide). *International Institute of Business Analysis (IIBA)*, 2009.

[15] N. Jalbert and W. Weimer. Automated duplicate detection for bug tracking systems. In *Proc. Int'l Conf. on Dependable Systems and Networks (DSN)*, pages 52–61, 2008.

[16] J. Li, H. Zhang, L. Zhu, R. Jeffery, Q. Wang, and M. Li. Preliminary results of a systematic review on requirements evolution. In *Proc. Int'l Conf. on Evaluation Assessment in Software Engineering (EASE)*, pages 12–21. IEEE, 2012.

[17] J. A. McCall, P. K. Richards, and G. F. Walters. Factors in software quality. In *Nat'l Tech.Information Service, no. Vol. 1, 2 and 3*. 1977.

[18] A. Mockus, R. T. Fielding, and J. D. Herbsleb. Two case studies of open source software development: Apache and mozilla. *ACM Trans. Softw. Eng. Methodol.*, 11(3):309–346, July 2002.

[19] J. Noll. Requirements acquisition in open source development: Firefox 2.0. In *OSS*, volume 275 of *IFIP*, pages 69–79. Springer, 2008.

[20] J. Noll and W.-M. Liu. Requirements elicitation in open source software development: a case study. In *Proc. Int'l Workshop on Emerging Trends in Free/Libre/Open Source Software Research and Development (FLOSS)*, pages 35–40. ACM, 2010.

[21] J. Robertson and S. Robertson. Volere: Requirements specification template. Technical report, Technical Report Edition 6.1, Atlantic Systems Guild, 2000.

[22] P. Runeson, M. Alexandersson, and O. Nyholm. Detection of duplicate defect reports using natural language processing. In *Proc. Int'l Conf. on Software Engineering (ICSE)*, pages 499–510. IEEE, 2007.

[23] R. J. Sandusky, L. Gasser, and G. Ripoche. Bug report networks: Varieties, strategies, and impacts in a f/oss development community. In *Proc. Int'l Workshop on Mining Software Repositories (MSR)*, pages 80–84, 2004.

[24] W. Scacchi. Understanding the requirements for developing open source software systems. In *IEE Proceedings - Software*, pages 24–39, 2001.

[25] C. Sun, D. Lo, X. Wang, J. Jiang, and S.-C. Khoo. A discriminative model approach for accurate duplicate bug report retrieval. In *Proc. Int'l Conf. on Software Engineering (ICSE)*, pages 45–54, 2010.

[26] Y. Tian, C. Sun, and D. Lo. Improved duplicate bug report identification. In *Proc. European Conf. on Software Maintenance and Reengineering (CSMR)*, pages 385–390. IEEE, 2012.

[27] R. Vlas and W. Robinson. A rule-based natural language technique for requirements discovery and classification in open-source software development projects. In *44th Hawaii International Conference on System Sciences (HICSS)*, pages 1–10, 2011.

[28] X. Wang, L. Zhang, T. Xie, J. Anvik, and J. Sun. An approach to detecting duplicate bug reports using natural language and execution information. In *Proc. Int'l Conf. on Software Engineering (ICSE)*, pages 461–470. ACM, 2008.

[29] A. Zaidman, M. Pinzger, and A. van Deursen. Software evolution. In P. A. Laplante, editor, *Encyclopedia of Software Engineering*, pages 1127–1137. Taylor & Francis, 2010.